COMPARISON STUDY ON SORTING TECHNIQUES IN STATIC DATA STRUCTURE

ANWAR NASER FRAK

A dissertation submitted in
partial fulfilment of the requirement for the award of the Degree of Master of
Computer Science (Software Engineering)

Faculty of Computer Science and Information Technology
Universiti Tun Hussein Onn Malaysia

MARCH 2016

To my beloved father and mother

This dissertation is dedicated to my father, who taught me that the best kind of knowledge to have is that which is learned for its own sake. It is also dedicated to my beloved mother, who taught me that even the largest task can be accomplished if it is done one step at a time.

Thank you for your love and support

ACKNOWLEDGMENT

First and foremost, I would like to thank the almighty God (ALLAH S.W.T) for all the abilities and opportunities He provided me through all my life and His blessings that enables me to do this research.

It is the greatest honor to be able to work under supervision of Dr. Mohd Zainuri Bin Saringat. I am grateful to be one of the luckiest persons who had a chance to work with him. I am thankful and gratified for all of his help, assistance, inspiration and guidance on the all aspects beside his patience and understanding

I wish to express my sincere gratitude to everyone who contributed to the successful completion of my study. I would like to express my gratitude to Universiti Tun Hussein Onn Malaysia (UTHM) for all support through my master.

ABSTRACT

To manage and organize large data is imperative in order to formulate the data analysis and data processing efficiency. Thus, to handle large data becomes highly enviable, whilst, it is premised that the sorting techniques eliminate ambiguities with less effort. Therefore, this study investigates the functionality of a set of sorting techniques to observe which technique to provide better efficiency in terms of sorting data. Therefore, five types of sorting techniques of static data structure, namely: Bubble, Insertion, Selection in group O (n^2) complexity and Merge, Quick in group O (n log n) complexity using the C++ programming language have been used. Each sorting technique was tested on four groups between 100 and 30000 of dataset. To validate the performance of sorting techniques, three performance metrics which are time complexity, execution time (run time) and size of dataset were used. All experimental setups were accomplished using simple linear regression where experimental results illustrate that Quick sort is more efficiency than Merge Insertion, Selection and Bubble sort based on run time and size of data using array and Selection sort is more efficient than Bubble and Insertion in large data size using array. In addition, Bubble, Insertion and Selection have good performance for small data size using array while Merge and Quick sort have good performance in large data size using array and sorting technique with good behavior O (n log n) more efficient rather than sorting technique with bad behavior is O (n^2) using array.

ABSTRAK

Mengurus dan mengatur kuantiti data yang banyak adalah penting untuk merumuskan analisis data dan kecekapan proses data. Dengan itu, untuk mengendalikan kuantiti data yang banyak, adalah lebih baik sekiranya teknik susunan dapat menghapuskan kesamaran dengan usaha yang minimum. Oleh itu, kajian ini menyiasat satu fungsi set teknik susunan untuk memerhatikan teknik mana yang dapat menyediakan kecekapan yang lebih baik. Kerana itu, lima jenis teknik susun struktur data statik, iaitu: Bubble, Insertion, Selection dalam kumpulan O (n^2) manakala Merge dan Quick dalam kumpulan O (n log n) yang menggunakan bahasa C++ telah pun digunakan. Setiap teknik susunan telah diuji dalam empat kumpulan di antara 100 dan 30000 data. Untuk mengesahkan prestasi dari teknik penyusunan, tiga metrik telah digunakan iaitu kerumitan dalam masa, pelaksanaan masa dan saiz data. Semua persediaan eksperimen telah pun selesai menggunakan regresi linear sederhana. Keputusan eksperimen menunujukkan bahawa Quick dan Merge adalah lebih cekap daripada Insertion, Selection dan Bubble adalah lebih cekap berdasarkan masa dan saiz data menggunakan tatasusunan dan Selection adalah lebih cekap daripada Bubble dan Insertion pada saiz data besar menggunakan tatasusunan. Di samping itu, Bubble, Insertion dan Selection mempunyai prestasi yang baik untuk saiz data kecil menggunakan tatatsusunan manakala Merge dan Quick mempunyai prestasi yang baik dalam saiz data yang besar menggunakan tatasusunan dan teknik susunan dengan tingkah laku O (n log n) yang lebih cekap daripada teknik susunan dengan tingkah laku adalah O (n^2) menggunakan tatasusunan.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

1. 1 Background of Study

Data structure is one of the most important techniques for organizing large data. It is considered as a method to systematically manage data in a computer that leads towards efficiently implementing different data types to make it suitable for various applications (Chhajed *et al.,* 2013). In addition, data structure is considered as a key and essential factor when designing effective and efficient algorithms (Okun *et al*., 2011; Black, 2009). There are various studies that demonstrate the importance of data structures in software design (Yang *et al.,* 2011; Andres *et al*., 2010).

So far, several researchers have focused on how to inscribe and improve the algorithm and ignoring data structures, while the data structures significantly affect the performance and the efficiency of the algorithm (Al-Kharabsheh *et al.,* 2013). Sorting is one of the basic function of computer processes which has been widely used on database systems. A sorting algorithm is the arrangement of data elements in some order either in ascending or descending order. The algorithm can also be helpful to group the data on the basis of a certain requirement such as Postal code or in the classification of data into certain age groups. Thus, sorting is a rearrangement of items in a requested list dedicated in order to produce solution for a desired result. A number of sorting
algorithms have been developed such as Quick sort, Merge sort, Insertion sort and Selection sort (Andres *et al*., 2010).

Meanwhile, several efforts have been taken to improve sorting techniques like Merge sort, Bubble sort, Insertion sort, Quick sort, Selection sort, each of them has a

different mechanism to reorder elements which increase the performance and efficiency of the practical applications and reduce the time complexity of each one. It is worth noting that when various sorting algorithms are being compared, there are afew parameters that must be taken into consideration, such as complexity, and execution time. The complexity is determined by the time taken for executing the algorithm (Goodrich, Tamassia & Mount, 2007). In general, the time complexity of an algorithm is generally written in the form of Big O(n) notation, where O represents the complexity of the algorithm and the value n represents the number of elementary operations performed by the algorithm (Jadoon *et al.,* 2011).

Hence, this study investigates the efficiency of five sorting techniques, namely Selection sort, Insertion sort, Bubble sort, Quick sort, Merge sort and their behaviour on small and large data set. To accomplish these major tasks, proposed methodology comprises of three phases are introduced implementation of sorting technique, calculation of their complexity and comparative analysis. Each phase contains different steps and delivers useful results to be used in the next phase. After that, performance of these five sorting techniques were evaluated by three performance measures which are time complexity, execution time (run time) and size of dataset used.

1.2 Motivation

Despite the importance of data organization, sorting a list of input numbers or character is one of the fundamental issues in computer science. Sorting technique attracted a great deal of research, for efficiency, practicality, performance, complexity and type of data structures (Gurram & Jaideep, 2011) Therefore, data management needs to involved a certain sorting process (Liu and Yang, 2013). As a result, sorting is an important part in the data organization. Many researchers are attentive in writing the sorting algorithms but did not focus on the type of data structure used on them. Finding the most efficient sorting technique involves in examining and testing these techniques to finish the main task as soon as possible and identifying the most suitable structure for fast sorting and study the factors that affect the practical performance of each algorithm in terms of its overall run time. Thus, the aim of this study is to evaluate the efficiency of five sorting algorithms which are Bubble sort, Insertion sort,

Selection sort, Merge sort and Quick sort using static array data structure and to compare and analyse the results based on their runtime and complexity.

1.3 Research Objectives

Based on the research background, the three objectives of this research are as follows:

i. implement five sorting techniques, namely Bubble sort, Insertion sort, Selection sort, Merge sort and Quick sort using static array data structure
ii. calculate the complexity of the implemented sorting techniques as in (i)
iii. compare and analyse result on the time taken and efficiency for the five sorting techniques based on the algorithms' complexity.

1.4 Research Scope

This thesis focuses only on testing the effectiveness of five different sorting techniques, namely Bubble sort, Insertion sort, Selection sort, Merge sort and Quick on four groups of datasets with various data sizes which are 100 to 1,000 (Group 1), 2,000 to 10,000 (Group 2), 11,000 to 20,000 (Group 3) and 21,000 to 30,000 (Group 4). The performance of these five sorting techniques is evaluated by three performance measures which are time complexity, execution time (run time) and size of datasets.

1.5 Thesis Outline

This thesis outlines the background study of this research project, focusing on five different sorting algorithms. The motivation and the scope of the research are also presented. Chapter 2 discusses the literature review on static array data structure and sorting techniques. Chapter 3 presented the methodology of this research while Chapter 4 explains the implementation and detailed steps of the work. Finally, Chapter 5 concludes with an elaboration of the research achievements and future work.

CHAPTER 2

LITERATURE REVIEW

2.1 Overview

The overall goal of this chapter is to establish the significance of the general field of study. First, an overview of data structures with its main activities are presented and followed by the description of sorting techniques. It also covers some evaluation measures such as execution time, algorithm complexity and the use of Least Squares Regression for finding estimation values among various groups and ends with a summary of the chapter.

2.2 Data Structures

Data structure is a systematic organization of information and data to enable it to be used effectively and efficiently especially when managing large data. Data structures can also be used to determine the complexity of operations and considered a way to manage large amounts of data such as the index of internet and large corporate databases (Chhajed *et al.*, 2013). According to Okun *et al*. (2011), an effective and efficient algorithm is required when designing highly efficient data structures. In the realm of software design, there are several studies that have attested the importance of data structures (Yang *et al.*, 2011; Andres *et al.*, 2010; Okun *et al.*, 2011).

Data structures are generally based on the ability of a computer to fetch and store data at any place in its memory, specified by a pointer with a bit stringer

presenting a memory address. Thus, the data structures are based on computing the addresses of data items with arithmetic operations, (Okun, et at., 2011).

Furthermore, there are two types of data structures which are static data structure such as array and dynamic data structure such as linked list. Static data structure has a fixed size and the elements of static data structures have fixed locations. But in dynamic data structures, the elements are added dynamically. Therefore, the locations of elements are dynamic and determined at runtime (Yang *et al.*, 2011).

2.3 Static Array Data Structures

An array is the arrangement of data in the form of rows and columns that is used to represent different elements through a single name but different indicators, thus, it can be accessed by any element through the index (Andres *et al.*, 2010). Arrays are useful in supplying an orderly structure which allows users to store large amounts of data efficiently. For example, the content of an array may be changed during runtime whereas the internal structure and the number of the elements are fixed and stable (Yang *et al.*, 2011).

An array could be called fixed array because they are not changed structurally after they are created. This means that the user cannot add or delete to its memory locations (making the array having less or more cells), but can modify the data it contains because it is not change structurally. Andres *et al.*, (2010) illustrated that there are three ways in which the elements of an array can be indexed.

i. zero-based index. It is the first element of the array which is indexed by subscript 0.
ii. one-based indexing. It is the first element of the array which is indexed by the subscript 1.
iii. n-based indexing. It is the base index of an array which can be freely chosen.

Therefore, arrays are important structures in the computer science, because they can store a large amount of data in a proper manner while comparing with the list structure, which are hard to keep track and do not have indexing capabilities that makes it weaker in terms of structure (Yang *et al.*, 2011).

2.3.1 Advantages of using Array

Each data structure has the weakness points and strength points. Listed below are advantages of array as mentioned by Andres *et al.* (2010):

i. arrays allow faster access to any item by using the index.
ii. arrays are simple to understand and use.
iii. arrays are very useful when working with sequences of the same type of data.
iv. arrays can be used to represent multiple data items of same type by using only single name but different indexes.

2.3.2 Disadvantages of using Array

Even though arrays are very useful data structures, however, there are some disadvantages as mentioned by Andres *et al.* (2010) which are listed below:

i. array items are stored in neighbouring memory locations, sometimes there may not be enough memory locations available in the neighbourhood.
ii. the elements of array are stored in consecutive memory locations. Therefore, operations like add, delete and swap can be very difficult and time consuming.

2.4 Execution Time

Execution time is the time taken to hold processes during the running of a program. The speed of the implementation of any program depends on the complexity of a technique or algorithm. If the complexity is low, then the implementation is faster, whereas when the complexity is high then the implementation is slow (Puschner & Koza, 1989). Keller (2000) argues that the execution time is the time for a program to process a given input. Keller believes that time is one of the important computer resources for two reasons: the time spent for the solution and the time spent for program implementation and for providing services.

2.5 Running Time Analysis

Running time is a theoretical process to calculate the approximate running time of a technique. For example, a program can take seconds or hours to complete an execution, depending on a particular technique used to lead the program (Bharadwaj & Mishra, 2013). Moreover, the runtime of a program describes the number of operations it executes during implementation and also the execution time of a technique. Furthermore, it should look forward to the worst case, an average case and best case performance of the technique. These definitions support the understanding of techniques complexity as mentioned by Bharadwaj & Mishra (2013).

2.6 Time Complexity

According to Estakhr (2013), the time complexity of an algorithm quantifies the amount of time taken by an algorithm to run as a function with the length of a string representing the input. The time complexity of an algorithm is commonly expressed using Big(O) notation, which excludes coefficients and lower order terms. When expressed this way, the time complexity is said to be described asymptotically as the input size goes to infinity. The time complexity is commonly estimated by counting the number of elementary operations performed by the algorithm, where an elementary operation takes a fixed amount of time to perform. Thus, the amount of time taken and the number of elementary operations performed by the algorithm differ by at most a constant factor (Michael, 2006).

Time can mean the number of memory accesses performed, the number of comparisons between integers, the number of times some inner loop is executed, or some other natural unit related to the amount of real time the algorithm will take. The research tries to keep this idea of time separated from clock time, since many factors unrelated to the algorithm itself can affect the real time such as the language used, type of computing hardware, the proficiency of the programmer and optimization used by the compiler. If the choice of the units is wise, all of the other factors will not matter to get an independent measure of the efficiency of the algorithm. The time complexities of the algorithms studied are shown in Table 2.1.

Table 2.1 :Time Complexity of Sorting Algorithms

Algorithm	Time complexity		
	Best case	Average case	Worst case
Bubble Sort	O(n)	$O(n^2)$	$O(n^2)$
Insertion Sort	O(n)	$O(n^2)$	$O(n^2)$
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Quick Sort	$O(n^2)$	$O(n^2)$	O(n.log (n))
Merge sort	O(n.log (n))	O(n.log (n))	O(n.log (n))

2.6.1 Worst-Case Analysis

The worst case analysis anticipates the greatest amount of running time that an algorithm needed to solve a problem for any input of size n. The worst case running time of an algorithm gives us an upper bound on the computational complexity and also guarantees that the performance of an algorithm will not get worse (Szirmay & Márton, 1998).

2.6.2 Best-Case Analysis

The best case analysis expects the least running time the algorithm needed to solve a problem for any input of size n. The running time of an algorithm gives a lower bound on the computational complexity. Most of the analysts do not consider the best case performance of an algorithm because it is not useful (Szirmay & Márton, 1998).

2.6.3 Average Case Analysis

Average case analysis is the average amount of running time that an algorithm needed to solve a problem for any input of size n. Generally, the average case running time is considered approximately as bad as the worst case time. However, it is useful to check the performance of an algorithm if its behaviour is averaged over

all potential sets of input data. The average case analysis is much more difficult to carry out, requiring tedious process and typically requires considerable mathematical refinement that causes worst case analysis to become more prevalent (Papadimitriou, 2003).

2.7 Big-O Notation

Big-O notation is used to characterize upper bound of a function that states the maximum value of resources needed by an algorithm to do the execution (Knuth, 1976). According to Black (2007), Big(O) notation has two major fields of application namely mathematics and computer science. In mathematics, it is usually used to show how closely a finite series approximates a given function. In computer science, it is useful in the analysis of algorithms. There are two usages of Big (O) notation which are infinite asymptotic and infinitesimal asymptotic. This singularity is only in the application and not in precept; however, the formal definition for the Big(O) is the same for both cases, only with different limits for the function evidence.

Let f(n) and g(n) be functions that map positive integers to positive real numbers. Say that f(n) is O(g(n)) (or if (n) ∈ O (g (n))) if there exists a real constant $c > 0$ and there be an integer constant $n_0 \geq 1$ such that $f(n) \leq c.\ g(n)$ for every integer $n \geq n_0$ as shown in Figure 2.1.

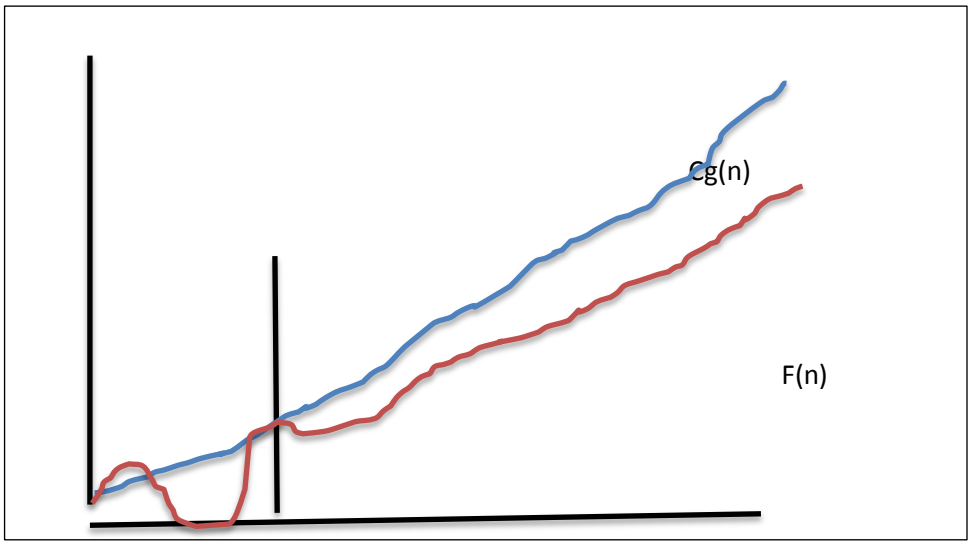

Figure 2.1: Big O Notation Graph.

REFERENCE

Al-Kharabsheh, K. S., AlTurani, I. M., AlTurani, A. M. I., & Zanoon, N. I. (2013). Review on Sorting Algorithms A Comparative Study. *International Journal of Computer Science and Security (IJCSS)*, *7*(3), 120.

Andres, B., Koethe, U., Kroeger, T., & Hamprecht, F. A. (2010). Runtime-flexible multi-dimensional arrays and views for C++ 98 and C++ 0x. *arXiv preprint arXiv:1008.2909*.

Astrachan, O. (2003, February). Bubble sort: an archaeological algorithmic analysis. In *ACM SIGCSE Bulletin* (Vol. 35, No. 1, pp. 1-5). ACM.

Batcher, K. E. (1968, April). Sorting networks and their applications. In*Proceedings of the April 30--May 2, 1968, spring joint computer conference* (pp. 307-314). ACM.

Bharadwaj, A., & Mishra, S. (2013). okun. *International Journal of Computer Applications*, *78*(14), 7-10.

Bishop, C. M. (2006). *Pattern recognition and machine learning*. springer.

Black, P. E. (2007). big-O notation. *Dictionary of Algorithms and Data Structures, 2007*.

Black, P. E. (2009). Dictionary of Algorithms and Data Structures, US National Institute of Standards and Technology

Chhajed, N., Uddin, I., & Bhatia, S. S. (2013). A Comparison Based Analysis of Four Different Types of Sorting Algorithms in Data Structures with Their Performances. *International Journal of Advance Research in Computer Science and Software Engineering*, *3*(2), 373-381.

Ching Kuang Shene1Michigan Technological University, Department of Computer Science Houghton A Comparative Study of Linkedlist Sorting Algorithms by, MI 49931-1295shene@mtu.edu 1996.

Coxon, A. P. M. (1999). *Sorting data: Collection and analysis* (Vol. 127). Sage Publications.

Estakhr, A. R. (2013). Physics of string of information at high speeds, time complexity and time dilation. *Bulletin of the American Physical Society*, *58*.

Goodrich, M., Tamassia, R., & Mount, D. (2007). *DATA STRUCTURES AND ALOGORITHMS IN C++*. John Wiley & Sons.

Gurram, H. K., & Jaideep, G. (2011). Index Sort. *International Journal of Experimental Algorithms (IJEA)*, *2*(2), 55-62.

Horan, P. K., & Wheeless, L. L. (1977). Quantitative single cell analysis and sorting. *Science*, *198*(4313), 149-157.

keller, Complexity. http://www.cs.hmc.edu

Knuth, D. E. (1976). Big omicron and big omega and big theta. *ACM Sigact News*, *8*(2), 18-24.

Liu, Y., & Yang, Y. 2013, December. Quick-merge sort algorithm based on Multi-core linux. In Mechatronic Sciences. Electric Engineering and Computer (MEC), International Conference on (pp. 1578-1583). IEEE

Mehlhorn, K. (2013). *Data structures and algorithms 1: Sorting and searching*(Vol. 1). Springer Science & Business Media.

Miss. Pooja K. Chhatwani, Miss. Jayashree S. Somani, 2013. Comparative Analysis & Performance of Different Sorting Algorithm in Data Structure Lecturer. Information Technology, Dr. N. P. Hirani Institute of Polytechnic. Pusad,Maharashtra, India

Okun, V., Delaitre, A., & Black, P. E. (2011). Report on the third static analysis tool exposition (SATE 2010). *vol. Special Publication*, 500-283.

Papadimitriou, C. H. (2003). *Computational complexity* (pp. 260-265). John Wiley and Sons Ltd..

Puschner, P., & Koza, C. (1989). Calculating the maximum execution time of real-time Tomita, E., Tanaka, A., & Takahashi, H. (2006). The worst-case time complexity for generating all maximal cliques and computational experiments.*Theoretical Computer Science*, *363*(1), 28-42.programs. *Real-Time Systems*, *1*(2), 159-176.

Sipser, Michael (2006). *Introduction to the Theory of Computation*. Course Technology Inc. ISBN 0-619-21764-2.

Szirmay-Kalos, L., & Márton, G. (1998). Worst-case versus average case complexity of ray-shooting. *Computing*, *61*(2), 103-131.

Waegeman, W., De Baets, B., & Boullart, L. (2008). ROC analysis in ordinal regression learning. *Pattern Recognition Letters*, *29*(1), 1-9.

Yang, Y., Yu, P., & Gan, Y. (2011, July). Experimental study on the five sort algorithms. In *Mechanic Automation and Control Engineering (MACE), 2011 Second International Conference on* (pp. 1314-1317). IEEE